**实验6：单向链表（基础版）**

**姓名\_李飞飞\_ 班级\_软工2206\_ 学号\_202105710309\_**

* **请阅读此说明：实验6满分100分；做完实验后请按要求将代码和截图贴入该文档。然后将此文档、源代码文件（.hpp, .cpp）打包上传到学习通。**

**1、（总分44分）链表基本构建和访问：**

**单向链表的基本结点类可以设计成：**

#pragma once

typedef int dataType; //处理的数据类型以int为例

using namespace std;

class node {

public:

    node(dataType d, node\* ptr = nullptr) { data = d; next = ptr; }

    dataType data;

    node\* next;

};

**根据课堂学习的链表初级篇相关的知识点，设计函数完成以下工作，原型分别为：**

1. **链表的正向构建（5分）：node\* buildListForward();**

**//从键盘输入若干数据，正向构建链表，返回表首。**

1. **链表的逆向构建（5分）：node\* buildListReverse();**

**//从键盘输入若干数据，逆向构建链表，返回表首。**

1. **链表的访问（5分）：void display(node\* xhead);**

**//根据表首依次访问链表结点数据，数据输出使用逗号分隔。**

1. **链表的改进查找（5分）：void search(node\* xhead, dataType value, node\*& p, node\*& pre);**

**//根据目标值value，在给定表首的链表中查找value的位置p以及前驱点pre**

1. **链表的删除（5分）：void deleteList(node\*& xhead);**

**//根据给定的表首回收链表的所有结点**

1. **链表的添加到尾（5分）：void add2tail(node\*& xhead, node\*& xtail, dataType value);**

**//根据给定的value，申请新结点并将其添加到链表的尾部**

1. **链表的添加到首（5分）：void add2head(node\*& xhead, node\*& xtail, dataType value);**

**//根据给定的value，申请新结点并将其添加到链表的首部**

1. **链表的尾部查找（5分）：node\* findListTail(node\* head);**

**//根据表首找到链表的最后一个结点（尾部）并返回**

1. **思考（4分）：为什么与链表相关的指针参数在3）设计形参为node\* ,而5) 6) 7)形参设计为node\*& ？**

**1）2）设计返回表首的目的是什么?还可以如何调整这种设计？请给出可能的设计原型。**

**请完成1）-8）的函数设计并通过测试程序list\_demo1.cpp：**

**实验解答：**

**❶黏贴1)-8)的函数的实现代码：**

**1）链表的正向构建（5分）：node\* buildListForward();**

**//从键盘输入若干数据，正向构建链表，返回表首。**

**1）实现：**

node\* buildListForward()//从键盘输入若干数据，正向构建链表，返回表首。

{

    dataType xdata;//接受输入的临时变量

    cin >> xdata;

    node\* head = new node(xdata), \* tail = head, \* Nnode; //建立第一个结点，表首和表尾指示同一个结点

    char c;

    while ((cin >> xdata).get(c)) //使用 cin 构建循环

    {

        Nnode = new node(xdata); //创建新结点空间

        tail->next = Nnode; //将新结点挂接在表尾

        tail = Nnode; //新结点成为新表尾

        if (c == '\n') break;

    }

    return head;

}

**2）链表的逆向构建（5分）：node\* buildListReverse();**

**//从键盘输入若干数据，逆向构建链表，返回表首。**

**2）实现：**

node\* buildListReverse()//从键盘输入若干数据，逆向构建链表，返回表首。

{

    dataType xdata;

    node\* head = NULL, \* Nnode; //建立第一个结点，表首和表尾指示同一个结点

    char c;

    while ((cin >> xdata).get(c)) //使用 cin 构建循环，当输入为非 dataType 类型或者 Ctrl+Z 结束

    {

        Nnode = new node(xdata); //创建新结点空间

        Nnode->next = head;

        head = Nnode; //新结点成为新表首

        if (c == '\n') break;

    }

    return head;

}

**3）链表的访问（5分）：void display(node\* xhead);**

**//根据表首依次访问链表结点数据，数据输出使用逗号分隔。**

**3）实现：**

void display(node\* xhead)//根据表首依次访问链表结点数据，数据输出使用逗号分隔。

{

    node\* p = xhead; //定义指针变量p指向表首

    if (p != NULL)

    {

        cout << p->data;

        p = p->next;

        while (p != NULL)//当p为空时，链表访问结束

        {

            cout << "," << p->data;

            p = p->next;//p指向下一个结点

        }

        cout << endl;

    }

}

**4）链表的改进查找（5分）：void search(node\* xhead, dataType value, node\*& p, node\*& pre);**

**//根据目标值value，在给定表首的链表中查找value的位置p以及前驱点pre**

**4）实现：**

void search(node\* xhead, dataType value, node\*& p, node\*& pre)//根据目标值value，在给定表首的链表中查找value的位置p以及前驱点pre

{

    p = xhead; //定义指针变量p指向表首

    pre = NULL;//定义指针变量pre指向空结点

    while (p != NULL && p->data != value)//当p为空或者p所指数据不等于目标值时，查找结束

    {

        pre = p;//将p前驱结点赋值给pre

        p = p->next;//p指向下一个结点

    }

}

**5）链表的删除（5分）：void deleteList(node\*& xhead);**

**//根据给定的表首回收链表的所有结点**

**5）实现：**

void deleteList(node\*& xhead)//根据给定的表首回收链表的所有结点

{

    node\* p = xhead, \* q; //定义指针变量p指向表首，q为临时指针

    while (p != NULL)//当p为空时，链表访问结束

    {

        q = p;//将p结点赋值给q

        p = p->next;//p指向下一个结点

        delete q;//回收q所指结点空间

    }

    xhead = NULL;//同步更新表首

}

**6）链表的添加到尾（5分）：void add2tail(node\*& xhead, node\*& xtail, dataType value);**

**//根据给定的value，申请新结点并将其添加到链表的尾部**

**6）实现：**

void add2tail(node\*& xhead, node\*& xtail, dataType value)//根据给定的value，申请新结点并将其添加到链表的尾部

{

    node\* newNode = new node(value);

    if (!xhead) {

        xhead = newNode;

        xtail = newNode;

    }

    else {

        xtail->next = newNode;

        xtail = newNode;

    }

}

**7）链表的添加到首（5分）：void add2head(node\*& xhead, node\*& xtail, dataType value);**

**//根据给定的value，申请新结点并将其添加到链表的首部**

**7）实现：**

void add2head(node\*& xhead, node\*& xtail, dataType value)//根据给定的value，申请新结点并将其添加到链表的首部

{

    node\* Nnode = new node(value); //创建新结点空间

    Nnode->next = xhead;

    xhead = Nnode;//新结点成为新表首

}

**8）链表的尾部查找（5分）：node\* findListTail(node\* head);**

**//根据表首找到链表的最后一个结点（尾部）并返回**

**8）实现：**

node\* findListTail(node\* head)//根据表首找到链表的最后一个结点（尾部）并返回

{

    node\* p = head, \* q = NULL; //定义指针变量p指向表首，q为临时指针

    while (p != NULL)//当p为空时，链表访问结束

    {

        q = p;//将p结点赋值给q

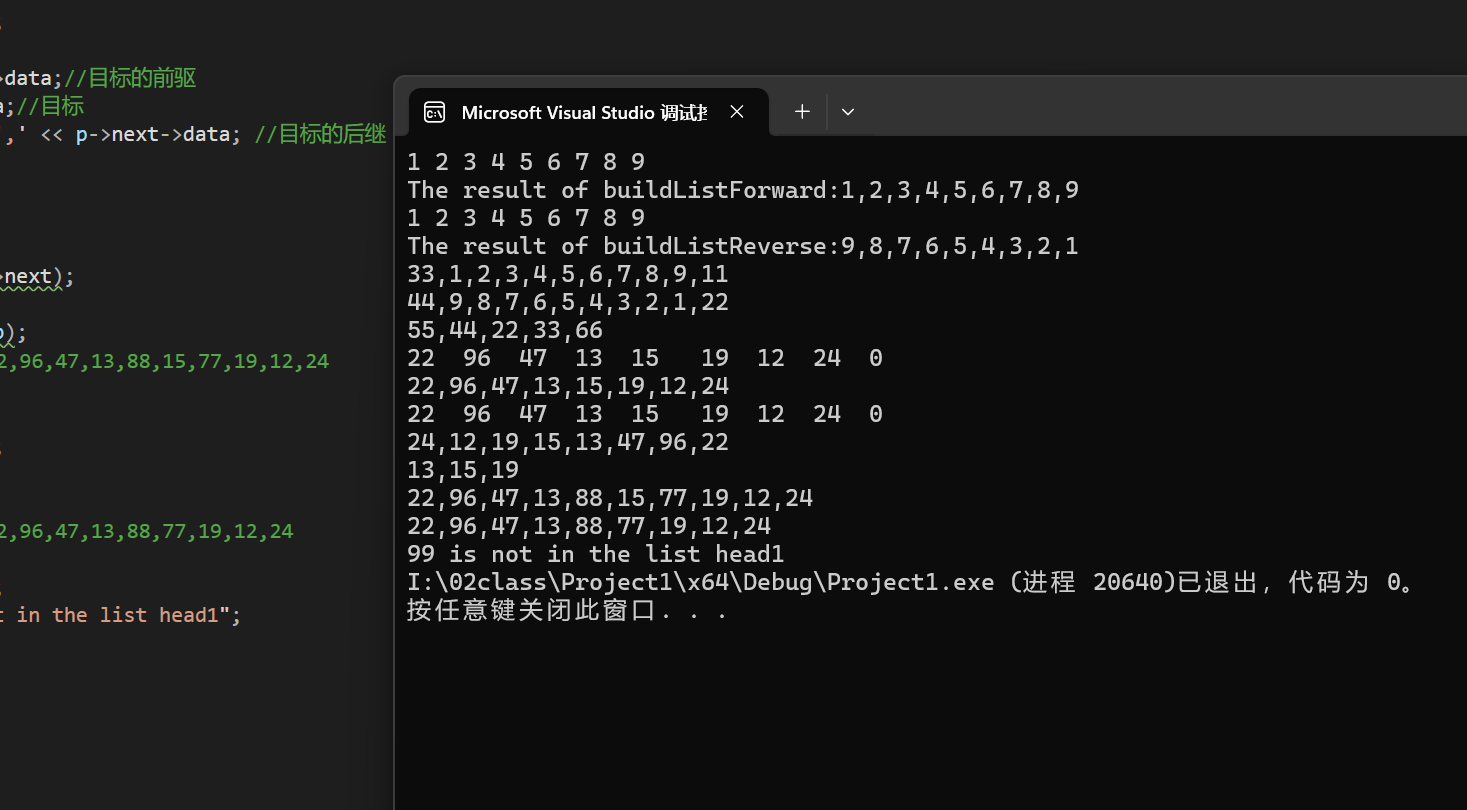
        p = p->next;//p指向下一个结点

    }

    return q;//返回表尾

}

**❷黏贴程序测试运行结果窗口（运行结果截屏）：**

****

1. **思考（4分）：为什么与链表相关的指针参数在3）设计形参为node\* ,而5) 6) 7)形参设计为node\*& ？**

**1）2）设计返回表首的目的是什么?还可以如何调整这种设计？请给出可能的设计原型。**

**答：**

**Node\*返回一个指针，因为在此方法中没有修改参数的需求所以不用传引用，若为node\*&是引用传参，可以通过函数修改此参数，在这样的方法中头尾节点可能被改变**

**2、（总分56分）链表的进阶工作（在第一个实验工作上继续）：**

**根据课堂学习的链表中级篇+部分高级篇相关的知识点，设计函数完成以下工作，原型分别为：**

1. **添加新值value到指定的p结点之后（6分）：void addAfterP(node \*& xhead, node\* p, dataType value);**

**//根据value值申请新结点并将其添加在p结点之后，如果p为空，则添加到表首。**

**//思考为什么这里设计需要表首的引用形参？**

**2）添加新值value到指定的p结点之前（6分）：void addBeforeP(node \*&xhead, node\* pre,node\* p, dataType value);**

**//根据value值申请新结点并将其添加在p结点之前，若p为空，则添加到表首。**

**3）删除链表指定位置p的结点（6分）：void deleteP(node\*& xhead,node\* pre,node\* p);**

**//根据给出的指定位置p以及前驱pre，删除xhead中的结点。**

**4）有序添加（6分）：void add2OrderedList(node \*&xhead, datatype value);**

**//根据value申请新结点，将其有序添加到有序表xhead中。**

**5）逆置链表（6分）：void reverseList(node \*&xhead);**

**//就地逆置链表**

**6）拷贝链表（6分）：node\* copyList(node\* xhead);**

**//按照给定xhead拷贝生成一张新链表，返回新链表的表首**

**7）基于链表的选择排序（6分）：void selectSortList(node \*xhead);**

**//使用选择排序策略将xhead整理为升序**

**8）基于链表的冒泡排序（7分）：void bubbleSortList(node \*xhead);**

**//使用冒泡排序策略将xhead整理为降序**

**9）删除链表内所有值为value的点（7分）：void deleteAllValues(node \*&xhead,dataType value);**

**//无回溯的策略。**

**//也可以使用search+deleteP实现带回溯的策略；void deleteAllValuesBacktrack(node \*&xhead,dataType value);**

**请完成1）-9）的函数设计并通过测试程序list\_demo2.cpp：**

**实验解答：**

**❶黏贴1)-9)的函数的实现代码：**

**1）添加新值value到指定的p之后（6分）：void addAfterP(node \*& xhead, node\* p,dataType value);**

**//根据value值申请新结点并将其添加在p之后，如果p为空，则添加到表首。**

**//思考为什么这里设计需要表首的引用形参？**

**1）实现：**

void addAfterP(node\*& xhead, node\* p, dataType value)

{

    node\* Nnode = new node(value);

    if (p == NULL)

    {

        Nnode->next = xhead;

        xhead = Nnode;

    }

    else

    {

        Nnode->next = p->next;

        p->next = Nnode;

    }

}

**2）添加新值value到指定的p之前（6分）：void addBeforeP(node \*&xhead, node\* p, dataType value);**

**//根据value值申请新结点并将其添加在p之前，若p为空，则添加到表首。**

**2）实现：**

void addBeforeP(node\*& xhead, node\* pre, node\* p, dataType value)

{

    node\* Nnode = new node(value, p);

    if (p == NULL)

    {

        Nnode->next = xhead;

        xhead = Nnode;

    }

    else

    {

        if (pre == NULL)

        {

            xhead = Nnode;

        }

        else  pre->next = Nnode;

    }

}

**3）删除链表指定位置p的结点（6分）：void deleteP(node\*& xhead,node\* pre,node\* p);**

**//根据给出的指定位置p以及前驱pre删除xhead中的结点。**

**3）实现：**

void deleteP(node\*& xhead, node\* pre, node\* p)

{

    if (p == xhead)

    {

        xhead = p->next;

        delete p;

    }

    else

    {

        pre->next = p->next;

        delete p;

        p = pre->next;

    }

}

**4）有序添加（6分）：void add2OrderedList(node \*&xhead, datatype value);**

**//根据value申请新结点，将其有序添加到有序表xhead中。**

**4）实现：**

void add2OrderedList(node\*& xhead, dataType value)

{

    node\* Nnode = new node(value);

    if (xhead == NULL)

    {

        xhead = Nnode;

    }

    else if (xhead->next == NULL)

    {

        if (xhead->data <= value)

        {

            xhead->next = Nnode;

        }

        else

        {

            Nnode->next = xhead;

            xhead = Nnode;

        }

    }

    else if (xhead->data > value)

    {

        Nnode->next = xhead;

        xhead = Nnode;

    }

    else

    {

        node\* pre = xhead;

        node\* p = xhead->next;

        while (p != NULL)

        {

            if (p->data > value)

            {

                Nnode->next = p;

                pre->next = Nnode;

                break;

            }

            p = p->next;

            pre = pre->next;

        }

        pre->next = Nnode;

    }

}

**5）逆置链表（6分）：void reverseList(node \*&xhead);**

**//就地逆置链表**

**5）实现：**

void reverseList(node\*& xhead)

{

    if (xhead != NULL && xhead->next != NULL)

    {

        node\* p = xhead->next;

        node\* pre = xhead;

        node\* s = p->next;

        while (p != NULL)

        {

            p->next = pre;

            pre = p;

            p = s;

            if (s != NULL) s = s->next;

        }

        xhead->next = NULL;

        xhead = pre;

    }

}

**6）拷贝链表（6分）：node\* copyList(node\* xhead);**

**//按照给定xhead拷贝生成一张新链表，返回新链表的表首**

**6）实现：**

node\* copyList(node\* xhead)

{

    if (xhead != NULL)

    {

        node\* head = new node(xhead->data);

        node\* tail = head;

        node\* p = xhead->next;

        while (p != NULL)

        {

            node\* Nnode = new node(p->data);

            tail->next = Nnode;

            tail = Nnode;

            p = p->next;

        }

        return head;

    }

    return NULL;

}

**7）基于链表的选择排序（6分）：void selectSortList(node \*xhead);**

**//使用选择排序策略将xhead整理为升序**

**7）实现：**

void selectSortList(node\* xhead)

{

    if (xhead != NULL)

    {

        node\* p, \* pre = xhead;

        while (pre->next != NULL)

        {

            node\* min = pre;

            p = pre->next;

            while (p != NULL)

            {

                if (p->data < min->data)

                {

                    min = p;

                }

                p = p->next;

            }

            dataType d = pre->data;

            pre->data = min->data;

            min->data = d;

            pre = pre->next;

        }

    }

}

**8）基于链表的冒泡排序（7分）：void bubbleSortList(node \*xhead);**

**//使用冒泡排序策略将xhead整理为降序**

**8）实现：**

void bubbleSortList(node\* xhead)

{

    if (xhead != NULL)

    {

        node\* p, \* pt;

        node\* pre = xhead;

        while (pre != NULL)

        {

            p = xhead;

            pt = p->next;

            while (pt != NULL)

            {

                if (p->data > pt->data)

                {

                    dataType d = p->data;

                    p->data = pt->data;

                    pt->data = d;

                }

                p = pt;

                pt = pt->next;

            }

            pre = pre->next;

        }

    }

}

**9）删除链表内所有值为value的点（7分）：void deleteAllValues(node \*&xhead,dataType value);**

**//无回溯的策略。**

**//也可以使用search+deleteP实现带回溯的策略；void deleteAllValuesBacktrack(node \*&xhead,dataType value);**

**9）实现：**

void deleteAllValues(node\*& xhead, dataType value)

{

    if (xhead != NULL)

    {

        node\* p = xhead;

        node\* pt = xhead->next;

        while (xhead->data == value)

        {

            delete p;

            xhead = pt;

            if (xhead == NULL) break;

            p = pt;

            pt = pt->next;

        }

        while (pt != NULL)

        {

            if (pt->data == value)

            {

                p->next = pt->next;

                delete pt;

                pt = p->next;

            }

            else

            {

                p = p->next;

                pt = pt->next;

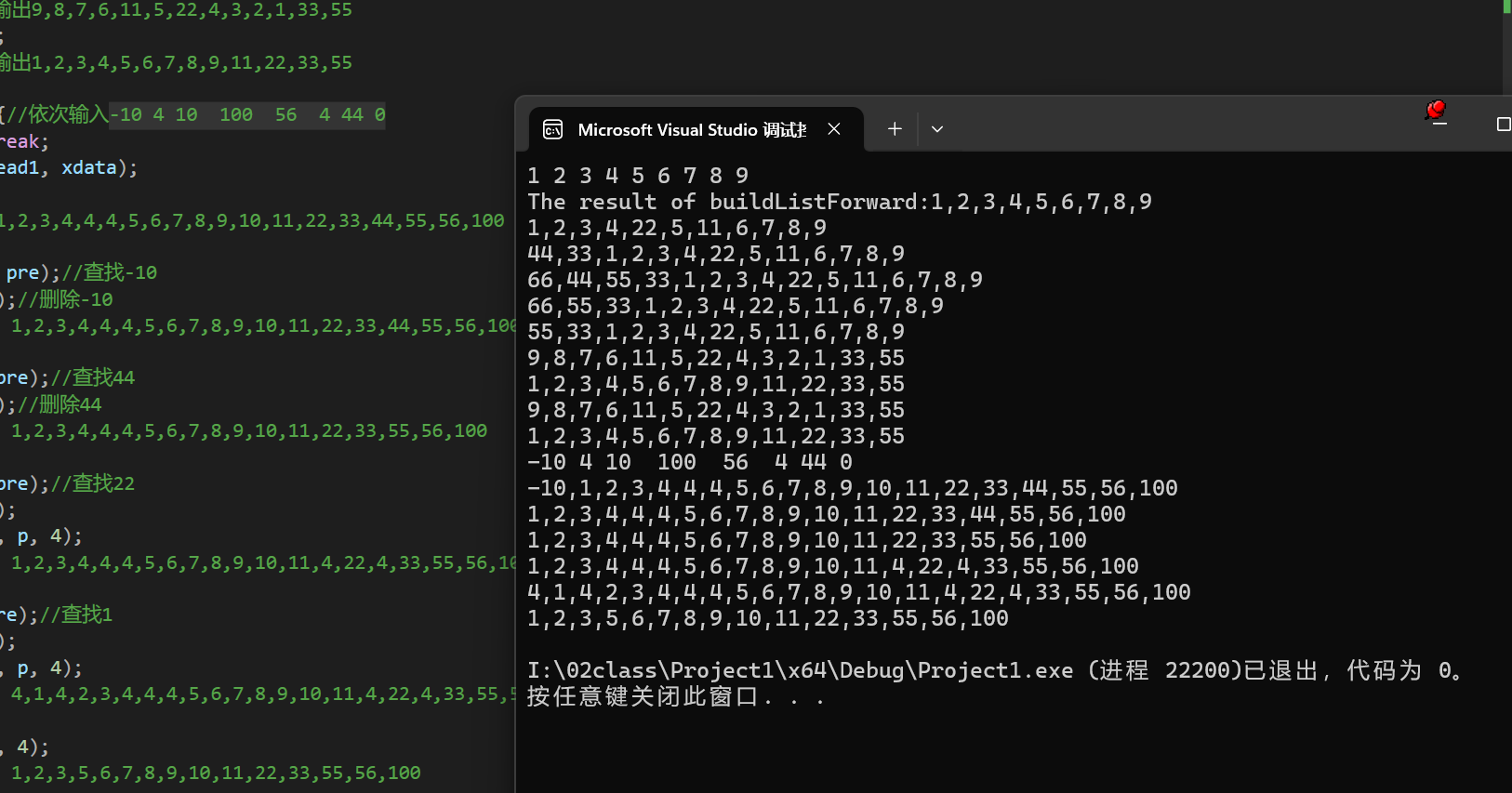
            }

        }

    }

}

**❷黏贴程序测试运行结果窗口（运行结果截屏）：**

****